UNITED STATES PATENT APPLICATION

FOR

METHOD AND APPARATUS FOR IMPLEMENTING A MULTIPLIER UTILIZING DIGITAL SIGNAL PROCESSOR BLOCK MEMORY EXTENSION

Attorney Docket No.: ALT.P030
Altera Docket No: A1252

Inventors: Asher Hazanchuk
Benjamin Esposito

Filed By:
Lawrence M. Cho
P.O. Box 2144
Champaign, IL 61825
(217) 377-2500

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number ER873666370US

Date of Deposit 04/22/2004

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to: Mail Stop Patent Application, Commissioner for Patents, P. O. Box 1450, Alexandra, VA 22313-1450

Lawrence M. Cho
(Typed or printed name of person mailing paper or fee)

L m. Ch
(Signature of person mailing paper or fee)

METHOD AND APPARATUS FOR IMPLEMENTING A MULTIPLIER UTILIZING DIGITAL SIGNAL PROCESSOR BLOCK MEMORY EXTENSION

TECHNICAL FIELD

[0001] The present invention relates to field programmable gate arrays (FPGAs) and the implementation of multipliers on FPGAs. More specifically, the present invention relates to a method and apparatus for implementing a multiplier utilizing digital signal processor (DSP) block memory extension.

BACKGROUND

[0002] FPGAs may be used to implement large systems that include millions of gates and megabits of embedded memory. Typical FPGAs include a plurality of programmable resources such as logic elements, memory blocks, and DSP blocks. When designing a multiplier on an FPGA, a traditional technique utilized the logic elements on the FPGA to build the multiplier. However, since a large number of logic elements were required to implement the multiplier, this technique resulted in producing a circuit having a large silicon size which was undesirable.

[0003] Other approaches used for designing a multiplier on an FPGA included using the DSP blocks on the FPGA. This technique produced a multiplier having a smaller silicon size which was an improvement over the traditional method of using logic elements. DSP blocks, however, have limited multiplier configuration flexibility. For example, a DSP block may support fixed configurations of only 9*9 bit, 18*18 bit, and 36*36 bit multiplication. When a multiplier has a configuration that exceeds a fixed configuration by even one bit, the next high resolution configuration would be required to implement that multiplier on the DSP block. Thus, this approach for multiplier design often resulted in unused multiplier resources on a DSP block. Survey results have indicated that when DSP blocks are used to implement multipliers on an FPGA, on average 75% of multiplier resources on the DSP blocks are unused. Since the number

of DSP blocks on an FPGA is limited, this approach was not efficient from a resource utilization standpoint.

[0004] Thus, what is needed is an effective and efficient method for implementing multipliers on an FPGA.

SUMMARY

[0005] Memory blocks on a FPGA are used to extend the resolution of a DSP block multiplier. For an n*m bit multiplier configuration, the DSP block is used to implement only a subset of the n*m bit multiplication functions. The remaining multiplication functions may be implemented by one or more memory blocks. A memory block may be used to store values that result from n*m bit multiplication functions not performed by the DSP block. Results generated by the DSP block and retrieved from the memory block may be scaled and summed together to generate a result equivalent to the n*m bit multiplication functions. The use of a DSP block memory extension reduces the amount of DSP block multiplier resources needed to implement a multiplier.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] The features and advantages of the present invention are illustrated by way of example and are by no means intended to limit the scope of the present invention to the particular embodiments shown, and in which:

[0007] Figure 1 illustrates a field programmable gate array (FPGA) that is configurable to implement a multiplier according to an embodiment of the present invention;

[0008] Figure 2 is a flow chart illustrating a method for implementing a multiplier on a FPGA according to an embodiment of the present invention;

[0009] Figure 3 illustrates a exemplary multiplier bit map frame according to an embodiment of the present invention;

[0010] Figure 4 is a flow chart illustrating a method for determining an offset for a decomposition product according to an embodiment of the present invention;

[0011] Figure 5 illustrates an exemplary multiplier bit map for a 10*12 multiplier according to an embodiment of the present invention;

[0012] Figure 6 illustrates an exemplary multiplier bit map for a 10*12 multiplier according to a second embodiment of the present invention.

[0013] Figure 7 illustrates an exemplary multiplier that may be implemented using the multiplier bit map shown in Figure 5 according to an embodiment of the present invention;

[0014] Figure 8 illustrates an exemplary multiplier that may be implemented using the multiplier bit map shown in Figure 5 according to a second embodiment of the present invention;

[0015] Figure 9 illustrates an exemplary multiplier that may be implemented using the multiplier bit map shown in Figure 6 according to an embodiment of the present invention;

[0016] Figure 10 illustrates an exemplary multiplier that may be implemented using the multiplier bit map shown in Figure 6 according to a second embodiment of the present invention.

DETAILED DESCRIPTION

[0017] Figure 1 illustrates an exemplary field programmable gate array (FPGA) 100 according to an embodiment of the present invention. According to one embodiment, the FPGA 100 is a chip having a hierarchical structure that may take advantage of wiring locality properties of circuits formed therein. The lowest level of the hierarchy is a logic element (LE) (not shown). An LE is a small unit of logic providing efficient implementation of user logic functions. According to one embodiment of the FPGA 100, an LE may include a 4-input lookup table (LUT) with a configurable flip-flop.

[0018] The FPGA 100 includes a plurality of logic-array blocks (LABs). Each LAB is formed from 10 LEs, LE carry chains, LAB control signals, LUT chain, and register chain connection lines. LUT chain connections transfer the output of one LE's LUT to the adjacent LE for fast sequential LUT connections within the same LAB. Register chain connection lines transfer the output of one LE's register to the adjacent LE's register within a LAB. LABs are grouped into rows and columns across the target device 100. A first column of LABs is shown as 110, a second column of LABs is shown as 111, and a third column of LABs is shown as 112.

[0019] The FPGA 100 includes memory blocks. The memory blocks may be, for example, dual port random access memory (RAM) blocks that provide dedicated true dual-port, simple dual-port, or single port memory up to various bits wide at up to various frequencies. The memory blocks may be grouped into columns across the target device in between selected LABs or located individually or in pairs within the FPGA 100. A column of memory blocks is shown as 120.

[0020] The FPGA 100 includes digital signal processing (DSP) blocks. The DSP blocks may be used to implement multipliers of various configurations with add or subtract features. The DSP blocks include shift registers, multipliers, adders, and accumulators. The DSP blocks may be grouped into columns across the FPGA 100. A column of DSP blocks is shown as 130.

[0021] The FPGA 100 includes a plurality of input/output elements (IOEs). Each IOE feeds an I/O pin (not shown) on the target device 100. The IOEs are located at the end of LAB rows and columns around the periphery of the FPGA 100. Each IOE includes a bidirectional I/O buffer and a plurality of registers for registering input, output, and output-enable signals. When used with dedicated clocks, the registers provide performance and interface support with external memory devices. A first column of IOEs are shown as 140, and a first row of IOEs are shown as 141.

[0022] The FPGA 100 includes LAB local interconnect lines (not shown) that transfer signals between LEs in the same LAB. The LAB local interconnect lines are driven by column and row interconnects and LE outputs within the same LAB. Neighboring LABs, memory blocks, or DSP blocks may also drive the LAB local interconnect lines through direct link connections.

[0023] The FPGA 100 also includes a plurality of row interconnect lines ("H-type wires") (not shown) that span fixed distances. Dedicated row interconnect lines route signals to and from LABs, DSP blocks, and memory blocks within the same row. The row interconnect lines span a distance of up to four, eight, and twenty-four LABs and are used for fast row connections in a four-LAB, eight-LAB, and twenty-four-LAB region. The row interconnects may drive and be driven by LABs, DSP blocks, RAM blocks, and horizontal IOEs.

[0024] The FPGA 100 also includes a plurality of column interconnect lines ("V-type wires") (not shown) that operate similarly to the row interconnect lines. The column interconnect lines vertically routes signals to and from LABs, memory blocks, DSP blocks, and IOEs. Each column of LABs is served by a dedicated column interconnect, which vertically routes signals to and from LABs, memory blocks, DSP blocks, and IOEs. These column interconnect lines include interconnects that traverse a distance of four, eight, and sixteen blocks in a vertical direction.

[0025] Figure 1 illustrates an exemplary embodiment of an FPGA. It should be appreciated that the FPGA may include components arranged in a manner different than that shown in Figure 1. An FPGA may also include components other than those described in reference to Figure 1. Thus, while the invention described herein may be utilized on the architecture described in Figure 1, it should be appreciated that it may also be utilized on different architectures, such as those employed by Altera® Corporation in its APEX™, and Mercury™ family of chips and those employed by Xilinx®, Inc. in its Virtex™ and Virtex™ II line of chips.

[0026] Components on the FPGA 100 may be used to implement an n*m bit multiplier. The multiplier may include a DSP block and one or more memory blocks in the FPGA 100. In one embodiment, the DSP block is used to generate a product (a first decomposition product) by multiplying a first plurality of the n bits from a first number and a first plurality of m bits from a second number. The memory block is used to store values that represent all the combinations of multiplying a second plurality of the n bits from the first number and a second plurality of the m bits from the second number. The memory block may be a look up table that returns a stored value (a second decomposition product) that represents a product of the second plurality of n bits from the first number and the second plurality of m bits from the second number.

[0027] The product from the DSP block may be scaled with respect to a position of the first plurality of bits from the first number and a position of the first plurality of bits from the second number. The stored value from the memory block may be scaled with respect to a position of the second plurality of bits from the second number and a position of the second plurality of bits from the second number. In an embodiment where the first and second plurality of bits from the first number and the first and second plurality of bits from the second number represent all the bits from the first and second number, the scaled product and the scaled stored value may be summed to generate a result that is equivalent to a product of the first and second number. In an embodiment where the first and second plurality of bits from the first number and the first and second plurality of bits from the second number do not represent all the bits from the first and

second number, additional memory blocks may be used to generate additional decomposition products that may be scaled and summed to generate a result equivalent to the product of the first and second number.

[0028] Figure 2 is a flow chart illustrating a method for implementing a multiplier on a FPGA according to an embodiment of the present invention. At 201, a DSP configuration is selected. The DSP configuration determines the number of bits that are to be supported by a DSP block for multiplication. In one embodiment, the largest dimension supported by the DSP block that is under the desired multiplier configuration is selected. It should be appreciated that if the largest dimension supported by the DSP block that is under the desired multiplier configuration is not available, a smaller dimension supported by the DSP block may also be selected. The DSP block may be configured to generate a product by performing multiplication on a first plurality of bits from a first number and a first plurality of bits from a second number.

[0029] At 202, one or more memory resources are selected for memory extension. According to an embodiment of the present invention, a capacity of the memory extension is determined. The capacity of the memory extension is determined based upon the DSP configuration selected and the desired multiplier configuration. The type of the one or more memory resources selected for the memory extension may be determined based upon the capacity of the memory extension and the storage capacity of the available memory resources available on the FPGA. It should be appreciated that other resources, such as registers, may also be selected to facilitate extension.

[0030] At 203, resources are allocated for designated bits in the numbers to be multiplied. According to an embodiment of the present invention, it is determined which bits in the numbers to be multiplied are to be processed by the DSP block and which bits in the numbers to be multiplied are to have values representing their products stored in the memory resources. According to an embodiment of the present invention, allocating resources for designated bits in numbers to be multiplied may be achieved by utilizing a multiplier bit map.

[0031] Figure 3 illustrates an exemplary multiplier bit map frame 300 according to an embodiment of the present invention. The multiplier bit map frame 300 is for a m*n bit multiplier configuration. The multiplier bit map 300 may be used to map out the bit designation for a first number and a second number to be multiplied. The multiplier bit map frame 300 has a horizontal dimension of m bits and a vertical dimension of n bits. The first number may be mapped along the horizontal dimension of the multiplier bit map frame 300 with its most significant bit at the left most side of the horizontal dimension of the multiplier bit map frame 300 and its least significant bit at the right most side of the horizontal dimension of the multiplier bit map frame 300. The second number may be mapped along the vertical dimension of the multiplier bit map frame 300 with its most significant bit at the bottom most side of the multiplier bit map frame 300 and its least significant bit at the top most side of the vertical dimension of the multiplier bit map frame 300. According to one embodiment a global least significant bit (GLSB) is designated at the upper right corner 301 of the multiplier bit map frame 300 and a global most significant bit (GMSB) is designated at the lower left corner 302 of the multiplier bit map frame 300.

[0032] Referring back to Figure 2, at 204, the resources are grouped together to form a multiplier. According to an embodiment of the present invention, grouping together the resources determines a number of adders to implement in the multiplier.

[0033] At 205, offsets are determined for the decomposition products from the resources. Because decomposition products represent the products of subsets of bits from the first number and subsets of bits from the second number, the decomposition products need to be scaled to reflect their true values. Offsets may be used to scale the decomposition products. According to an embodiment of the present invention, an offset for a decomposition value may be determined from the position of its corresponding subset of bits from the first number and the position of its corresponding subset of bits from the second number.

[0034] Figure 4 is a flow chart illustrating a method for determining an offset for a decomposition product according to an embodiment of the present invention. The procedure illustrated may be used to implement 205 in Figure 2. At 401, a position for a global least significant bit is established on a multiplier bit map. According to an embodiment of the present invention, the position for the global least significant bit is the relative position on the multiplier bit map where the least significant bit of the numbers to be multiplied is directed. In one embodiment, the global least significant bit is established on the top right corner of the multiplier bit map.

[0035] At 402, a position for a local most significant bit is established for a resource on the multiplier bit map. According to an embodiment of the present invention, the position for the local most significant bit for a resource is a relative position on the resource where the least significant bit of the numbers to be processed or stored is directed. In one embodiment, the local least significant bit is established on the top right corner of the resource.

[0036] At 403, the vertical and horizontal distance between the global least significant bit on the multiplier bitmap and the local least significant bit for a resource is summed.

[0037] An example of implementing a 12*10 bit dimension multiplier on a FPGA is illustrated with reference to Figure 2 according to an embodiment of the present invention. At 201, a DSP configuration is selected. In an example where a DSP block may be configured to support the fixed configurations of only 9*9 bit, 18*18 bit, and 36*36 bit multiplication, the 9*9 bit configuration is selected.

[0038] At 202, memory resources are selected for memory extension. The capacity of the memory extension is determined based upon the DSP configuration selected and the multiplier configuration desired. For a 9*9 bit configuration selected on a DSP block for a 12*10 bit multiplier configuration, a memory extension is required to be able to store values representing combinations of multiplying a 3 bit binary number with a 10 bit binary number for a first

dimension. The memory extension is also required to be able to store values representing combinations of multiplying a 12 bit binary number with a 1 bit binary number for a second dimension. A first and second memory block each capable of storing 256*8 bits may be used to store values representing combinations of multiplying a 3 bit binary number with a 10 bit binary number for the one dimension and to store values representing combinations of multiplying a 3 bit binary number with a 1 bit binary number for the second dimension. A 9-bit register may be used to store values representing combinations of multiplying a 9 bit binary number with a 1 bit binary number for the second dimension. It should be appreciated that a third memory block may be implemented instead of the 9-bit register.

[0039] At 203, resources are allocated for designated bits in numbers to be multiplied. It is determined which of the bits in the numbers to be multiplied are to be processed by the DSP block and which bits in the numbers to be multiplied are to have values representing their products stored in the memory resources.

[0040] Figure 5 illustrates an exemplary multiplier bit map 500 for a 10*12 multiplier configuration according to a first embodiment of the present invention. As mapped on the multiplier bit map 500, a DSP block 510 that is configured to support 9*9 bit multiplication is designated to multiply the 9 most significant bits of a first number with the 9 most significant bits of the second number. A 9-bit register 520 is configured to latch the 9 most significant bits of the first number if the least significant bit of the second number is 1. A first memory block 530 is configured to store values that represent all the combinations of multiplying the 3 least significant bits of the first number with the 5 most significant bits of the second number. A second memory block 540 is configured to store values that represent all combinations of multiplying the 3 least significant bits of the first number with the 5 least significant bits of the second number.

[0041] Figure 6 illustrates an exemplary multiplier bit map 600 for a 10*12 multiplier configuration according to a second embodiment of the present invention. As mapped on the multiplier bit map 600, a DSP block 610 that is configured to support 9*9 bit multiplication is

designated to multiply the 9 least significant bits of a first number with the 9 least significant bits of the second number. A 9-bit register 620 is configured to latch the 9 least significant bits of the first number if the most significant bit of the second number is 1. A first memory block 630 is configured to store values that represent all the combinations of multiplying the 3 most significant bits of the first number with the 5 most significant bits of the second number. A second memory block 640 is configured to store values that represent all combinations of multiplying the 3 most significant bits of the first number with the 5 least significant bits of the second number.

[0042] Referring back to Figure 2, at 204, the resources are grouped together to form a multiplier. According to an embodiment of the present invention, grouping together the resources determines a number of adders to implement in the multiplier.

[0043] Figure 7 illustrates an exemplary multiplier 700 that may be implemented according to the multiplier bit map shown in Figure 5 according to an embodiment of the present invention. The multiplier 700 routes the output of the DSP block 510, the shift register 520, the first memory block 530, and the second memory block 540 to an adder 710.

[0044] Figure 8 illustrates an exemplary multiplier 800 that may be implemented according to the multiplier bit map shown in Figure 5 according to a second embodiment of the present invention. The multiplier 800 routes the output of the DSP block 510 and the output of the shift register 520 to a first adder 810. The multiplier 800 routes the output of the first memory block 530 and the output of the second memory block 540 to an adder 820. The multiplier 800 routes the output of the first and second adders 810 and 820 to a third adder 830.

[0045] Figure 9 illustrates an exemplary multiplier 900 that may be implemented according to the multiplier bit map shown in Figure 6 according to an embodiment of the present invention. The multiplier 900 routes the output of the DSP block 610, the shift register 620, the first memory block 630, and the second memory block 640 to an adder 910.

[0046] Figure 10 illustrates an exemplary multiplier 1000 that may be implemented according to the multiplier bit map shown in Figure 6 according to a second embodiment of the present invention. The multiplier 1000 routes the output of the DSP block 610 and the output of the register 620 to a first adder 1010. The multiplier 1000 routes the output of the first memory block 630 and the output of the second memory block 640 to an adder 1020. The multiplier 1000 routes the output of the first and second adders 1010 and 1020 to a third adder 1030.

[0047] At 205, offsets are determined for the decomposition products from the resources.

[0048] Referring back to Figure 5, the vertical and horizontal distances from the local least significant bit of DSP block 510 to the global least significant bit of the multiplier bit map 502 are 1 and 3, making the offset for the DSP block 510 4. The vertical and horizontal distances from the local least significant bit of shift register 520 to the global least significant bit of the multiplier bit map 502 are 0 and 3, making the offset for the shift register 520 3. The vertical and horizontal distances from the local least significant bit of the first memory block 530 to the global least significant bit of the multiplier bit map 502 are 5 and 0, making the offset for the first memory block 530 5. The vertical and horizontal distances from the local least significant bit of the second memory block 540 to the global least significant bit of the multiplier bit map 502 are 0 and 0, making the offset for the second memory block 540 0.

[0049] The offsets determined from the multiplier bit map 500 are indicated in the multipliers shown in Figures 7 and 8. Referring back to Figure 7, the output from DSP block 510 is to be scaled 4 bits to the left (4L). The output from the register 520 is to be scaled 3 bits to the left (3L). The output from the first memory block 530 is to be scaled 5 bits to the left (5L). The output from the second memory block 540 need not be scaled (0L). It should be appreciated that scaling the outputs from components 510, 520, 530, and 540 may be achieved by routing the outputs on routing resources 701-704, respectively to the adder 710 at inputs of appropriate

significance. In this embodiment, additional resources for shifting the bits of the outputs are not required.

[0050] Referring back to Figure 8, the output from DSP block 510 is similarly scaled 4 bits to the left (4L). The output from the shift register 520 is to be scaled 3 bits to the left (3L). The output from the first memory block 530 is to be scaled 5 bits to the left (5L). The output from the second memory block 540 need not be scaled (0L). It should be appreciated that scaling the outputs from components 510 and 520 may be achieved by routing the outputs on routing resources 801 and 802, respectively to the adder 810 at inputs of appropriate significance. Similarly, scaling the outputs from components 530 and 540 may be achieved by routing the outputs on routing resources 803 and 804 to the adder 820 at inputs of appropriate significance. In this embodiment, additional resources for shifting the bits of the outputs are not required.

[0051] Referring back to Figure 6, the vertical and horizontal distances from the local least significant bit of DSP block 610 to the global least significant bit of the multiplier bit map 602 are 0 and 0, making the offset for the DSP block 610 0. The vertical and horizontal distances from the local least significant bit of shift register 620 to the global least significant bit of the multiplier bit map 602 are 9 and 0, making the offset for the shift register 620 9. The vertical and horizontal distances from the local least significant bit of the first memory block 630 to the global least significant bit of the multiplier bit map 602 are 5 and 9, making the offset for the first memory block 630 14. The vertical and horizontal distances from the local least significant bit of the second memory block 640 to the global least significant bit of the multiplier bit map 602 are 0 and 9, making the offset for the second memory block 640 9.

[0052] The offsets determined from the multiplier bit map 600 are indicated in the multipliers shown in Figures 9 and 10. Referring back to Figure 9, the output from DSP block 610 is to be scaled 4 bits to the left (4L). The output from the shift register 620 is to be scaled 3 bits to the left (3L). The output from the first memory block 630 is to be scaled 5 bits to the left (5L). The output from the second memory block 640 need not be scaled (0L). It should be

appreciated that scaling the outputs from components 610, 620, 630, and 640 may be achieved by routing the outputs on routing resources 901-904, respectively to the adder 910 at inputs of appropriate significance. In this embodiment, additional resources for shifting the bits of the outputs are not required.

[0053] Referring back to Figure 10, the output from DSP block 610 is similarly scaled 4 bits to the left (4L). The output from the shift register 620 is to be scaled 3 bits to the left (3L). The output from the first memory block 630 is to be scaled 5 bits to the left (5L). The output from the second memory block 640 need not be scaled (0L). It should be appreciated that scaling the outputs from components 610 and 620 may be achieved by routing the outputs on routing resources 1001 and 1002, respectively to the adder 1010 at inputs of appropriate significance. Similarly, scaling the outputs from components 630 and 640 may be achieved by routing the outputs on routing resources 1003 and 1004 to the adder 1020 at inputs of appropriate significance. In this embodiment, additional resources for shifting the bits of the outputs are not required.

[0054] Figures 8 and 10 illustrate multipliers that fully scale decomposition products prior to performing a first stage of additions. It should be appreciated that where a multiplier has more than one stage of addition, the scaling may be divided among the available stages. For example, in Figure 8, the output from the DSP block 510 may be scaled 1 bit to the left (1L) and the output of the adder 810 may be scaled 3 bits to the left (3L).

[0055] Referring back to Figure 7, multiplier 700 may be used to multiply the numbers 100110010100 and 1000101101. DSP block 510 multiplies 101001100 with 100010110 to generate the product 10100110001001100, which is a first decomposition product. Shift register 520 shifts out a value 100110010, which is a second decomposition product. First memory block 530 transmits stored value 1000100, an equivalent value to the product of 100 and 10001, which is a third decomposition product. Second memory block 540 transmit stored value 110100, an equivalent value to the product of 100 and 01101, which is a fourth decomposition product. Line

701 routes the output from DSP block 510 to adder 710 such that the first decomposition product is scaled to 101001100010011000000. Line 702 routes the output from the register 520 to adder 710 such that the second decomposition product is scaled to 100110010000. Line 703 routes the output from the first memory block 530 to adder 710 such that the third decomposition product is scaled to 100010000000. Line 704 routes the output from the second memory block 540 to adder 710 such that the fourth decomposition product is scaled to 110100. The adder 710 sums the decomposition products to generate the value 10100110101011100000100.

[0056] Referring back to Figure 8, multiplier 800 may be used to multiply the numbers 100110010100 and 1000101101. DSP block 510 multiplies 101001100 with 100010110 to generate the product 10100110001001100, which is a first decomposition product. The register 520 shifts out a value 100110010, which is a second decomposition product. First memory block 530 transmits stored value 1000100, an equivalent value to the product of 100 and 10001, which is a third decomposition product. Second memory block 640 transmit stored value 110100, an equivalent value to the product of 100 and 01101, which is a fourth decomposition product. Line 801 routes the output from DSP block 510 to adder 810 such that the first decomposition product is scaled to 101001100010011000000. Line 802 routes the output from the register 520 to adder 810 such that the second decomposition product is scaled to 100110010000. Line 803 routes the output from the first memory block 530 to adder 820 such that the third decomposition product is scaled to 100010000000. Line 804 routes the output from the second memory block 540 to adder 810 such that the fourth decomposition product is scaled to 110100. The adder 810 sums the decomposition products to generate the value 10100110011100101000O. The adder 820 sums the decomposition products to generate the value 100010110100. Line 805 routes the output from adder 810 to the adder 830. Line 806 routes the output from adder 820 to the adder 830. The adder 830 sums the decomposition products to generate the value 10100110101011100000100.

[0057] Figures 2 and 4 are flow charts illustrating embodiments of the present invention. Some of the procedures illustrated in the figures may be performed sequentially, in parallel or in

an order other than that which is described. It should be appreciated that not all of the procedures described are required, that additional procedures may be added, and that some of the illustrated procedures may be substituted with other procedures.

[0058] Figures 5 and 6 illustrate exemplary multiplier bit maps for a 10*12 multiplier configuration according to embodiments of the present invention. It should be appreciated that other multiplier bit maps may be generated using the techniques described and that a different allocation of resources on the FPGA may be made to designated bits in the numbers to be multiplied. Figures 7 and 8 illustrate multipliers that may be implemented according to the multiplier bit map shown in Figure 5 and Figures 9 and 10 illustrate multipliers that may be implemented according to the multiplier bit map shown in Figure 6. It should be appreciated that the multipliers disclosed are exemplary multipliers and that other multipliers may be configured according to the multiplier bit maps shown in Figures 5 and 6.

[0059] The techniques described herein are not limited to any particular hardware or software configuration. They may find applicability in any computing or processing environment. The techniques may be implemented in hardware, software, or a combination of the two. The techniques may be implemented in programs executing on programmable machines such as mobile or stationary computers, and other electronic devices, that each include a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements). One of ordinary skill in the art may appreciate that the embodiments of the present invention can be practiced with various computer system configurations, including multiprocessor systems, minicomputers, mainframe computers, and other systems. The embodiments of the present invention can also be practiced in distributed computing environments where tasks may be performed by remote processing devices that are linked through a communications network.

[0060] Program instructions may be used to cause a general-purpose or special-purpose processing system that is programmed with the instructions to perform the operations described

herein. Alternatively, the operations may be performed by specific hardware components that contain hardwired logic for performing the operations, or by any combination of programmed computer components and custom hardware components. The methods described herein may be provided as a computer program product that may include a machine readable medium having stored thereon instructions that may be used to program a processing system or other electronic device to perform the methods. The term "machine readable medium" used herein shall include any medium that is capable of storing or encoding a sequence of instructions for execution by the machine and that cause the machine to perform any one of the methods described herein. The term "machine readable medium" shall accordingly include, but not be limited to, solid-state memories, optical and magnetic disks, and a carrier wave that encodes a data signal. Furthermore, it is common in the art to speak of software, in one form or another (e.g., program, procedure, process, application, module, logic, and so on) as taking an action or causing a result. Such expressions are merely a shorthand way of stating that the execution of the software by a processing system causes the processor to perform an action to produce a result.

[0061] In the foregoing specification, the embodiments of the present invention have been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the embodiments of the present invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than restrictive sense.